Università di Pisa
Dipartimento di Informatica

Technical Report: TR-01-24

Flocking by A Set of Autonomous Mobile Robots

Vincenzo Gervasi Giuseppe Prencipe

October 30, 2001

ADDRESS: Corso Italia 40, 56125 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

Flocking by A Set of Autonomous Mobile Robots*

Vincenzo Gervasi   Giuseppe Prencipe

October 30, 2001

1 Overview

Control and coordination of a set of autonomous vehicles that can freely move on a plane is a widely studied topic in robotics. The focus on this kind of problem has grown in recent years because of the increased interest in studying systems populated by many, simple units, instead of few, powerful ones. In particular, these units simply observe the environment by using their sensors, and react following simple rules: the reaction to the environmental stimuli is called the *behavior* of the unit. Despite the simplicity of the units, it has been shown that these groups of vehicles can exploit rather complex group behaviors [14]. Moreover, such a system is preferable to one made up of just one large and powerful unit for many reasons: low costs in changing faulty units; ability to solve tasks otherwise unsolvable by a single unit [6, 16] (e.g., robots that have to move big objects [15]); certain problems can be solved quicker [2] (e.g., robots that are asked to clean a room [13]); for fault tolerance considerations; the decreased cost through simpler individual unit design. An extensive survey can be found in [4].

One major question that arises is: How is it possible to properly coordinate these group of mobile robots, such that they can together accomplish what they are asked to do? And another question is: How simple can these units be [7]?

In this paper we study the *flocking problem*: a set of mobile units are required to follow a leader unit while keeping a predetermined formation (i.e., they are required to move in flock, like a group of soldiers). Moreover, the units in the flock do not know beforehand the path the leader will take: their task is just to follow him wherever he goes, and to keep the formation while moving. This is a problem often studied in robotics, with several applications (e.g. military [3], or in factories, where robots can be asked to move heavy loads). The approach usually adopted to study this and similar problems, is to design solutions based on heuristics and tailored on the capabilities of the robots employed, and then test them by computer simulations, or on real robots.

For instance, in [14], experiments are conducted on a team of simple mobile units in order to produce *complex* behaviors, by compounding *basic* ones (such as

*A preliminary version of this paper has been published in [11].

safe-wondering, i.e. the ability to avoid collisions while moving; dispersion, i.e. the ability of the robots to spread out over an area; aggregation, i.e. the ability to gather; and homing, i.e. the ability to reach a predetermined destination). In particular, the author points out that flocking can be obtained combining safe-wandering, aggregation, dispersion, and homing. Hence, in her experiments, all the units have a common destination to reach.

T. Balch and C. Arkin studied formation and navigation problems in multi-robot teams. In particular in [1] the problem of specifying the behavior for the navigation of a mobile unit is analyzed, and results of both computer simulation and real experimentation are reported. In [3] the approach is extended to multi-robot teams that navigate the environment maintaining particular formations: in particular the case of a line, column, diamond and wedge are examined. In their study, the authors assumed that the path along which the group of robots has to move is known in advance to every unit. This same assumption is made in [5], where the robots are asked to move in a matrix shape along a path represented by a straight line followed by a right turn and then a straight line again. In contrast, in this paper we do not assume any knowledge by the followers of the path that the leader will follow. The followers have only a common description of the formation they have to keep while moving.

A similar problem is studied in [17], where the author derives equations describing navigational strategies for robots moving in formation, and following the movement described by one (ore more) leader. In the studied framework, the robots have identities, hence their positions in the formation are fixed. Moreover, in order for the i-th robot to compute its position at time t, it has to know the position of either the $(i-1)$-th robot or the leader at time t. Hence, some degree of synchrony has to be introduced in order to implement these strategies.

In this paper, we analyze the flocking problem by using very simple units and by dropping the assumption that all the ships in the flock know the path, or that they can derive it (e.g. by observing the orientation of the leader's prow, or by deriving it by observing the leader in different positions). Following the motivations that prompted previous studies ([8, 12]), we adopt simple units to study the problem: the ships are completely anonymous, identical (no identities are used during the computation), asynchronous, memoryless, and with no means of direct communication. We describe an algorithm (the same for all the ships) that allows the followers to keep a formation given to them as input, while following a path determined at run-time by the leader, that acts completely independently and does not behave according to the followers' algorithm. Moreover, we present results of computer simulations that show the effectiveness of the proposed solution.

2 The Model

We consider[1] a system of autonomous mobile units (ships). There are two kinds of ships in the environment: the *leader* and the *followers*. The leader acts independently from the others, and we can assume that it is driven by a human pilot. In the following we will discuss only about the followers.

Each ship is capable of observing its surrounding, computing a destination based on what it observed, and moving towards the computed destination; hence it performs an (endless) cycle of observing, computing, and moving.

Each ship has its own *local view* of the world. This view includes a local Cartesian coordinate system having an origin (that without losing generality we can assume to be the position of the ship), a unit of length, and the *directions* of two coordinate axes (which we will refer to as the x and y axes), together with their *orientations*, identified as the positive and negative sides of the axes. In general, there is no agreement among the followers on the chirality of the local coordinate systems (i.e., the ships do not share the same concept of where North, East, South, and West are).

The ships are modeled as units with computational capabilities, which are able to freely move in the plane. They are equipped with sensors that let each ship observe the positions of the others with respect to their local coordinate system. Each ship is viewed as a point, and can see all the other ships in the flock (and the leader).

The ships act totally independently and *asynchronously* from each other, and do not rely on any centralized directives, nor on any common notion of time. Furthermore, they are *oblivious*, meaning that they do not (need to) remember any previous observation nor computations performed in the previous steps. Note that this feature gives the algorithms designed in this model the nice property of self-stabilization [9]: in fact, every decision taken by a follower can not depend on what happened in the system previously, and hence is not based on corrupted data stored in its local memory. They have as input, however, the same pattern $\mathbb{F}$ representing the flock to be kept. $\mathbb{F}$ is described as a set of coordinates in the plane, relative to a point representing the leader.

The ships in the flock are *anonymous*, meaning that they are a priori indistinguishable by their appearances, and they do not (need to) have any kind of identifiers that can be used during the computation. They can only distinguish if a ship is the leader or a fellow follower. Moreover, there are no explicit direct means of communication; hence the only way they have to acquire information from the fellow ships is by observing them[2].

They execute the same algorithm, which takes as input the observed positions of the ships, and returns a destination point towards which the executing ship moves. A ship, asynchronously and independently from the other ships,

[1] The model we adopt is based on one introduced in [10, 8]. It has been adapted, however, to the particular problem under study.

[2] The obliviousness of the ships also renders the observations weaker. In fact, nothing observed in the past can be remembered, hence used in order to let the ships organize themselves to accomplish their task.

(1) observes the environment (*Look*), by taking a snapshot of the positions of all other ships with respect to its local coordinate system. Each ship is viewed as a point, and therefore its position in the plane is given by its coordinates. The observation returns the positions of all the other ships in the plane (leader and followers). (2) It computes a destination point p according to its oblivious algorithm (*Compute*); the local computation is based only on the current (i.e., at the time of the previous Look) locations of the observed ships. (3) Finally, the ship moves an unpredictable amount of space towards p (*Move*), which is however assumed to be neither infinite, nor infinitesimally small (see Assumption A2 below), and goes back to the Look state. Hence, there is no assumption on the maximum distance a ship can travel before observing again (apart from the bound given from the destination point that has to be reached). The life of a follower consists in repeating an endless cycle of *states* (1)–(3). Moreover, the only assumptions made in the model are the following [10]: **(A1)** The time for a ship to complete a Look-Compute-Move cycle is neither infinite nor infinitesimally small (i.e., is finite and bounded from below). **(A2)** For each follower f, there exists an arbitrary (small) constant $\delta_f > 0$, representing the minimum distance it travels in the Move state; if the computed destination point is closer than δ_f, f will reach it. **(A3)** Since we need to model ships that "continuously" move, we assume that the time spent in looking and computing is negligible compared to the time spent in moving.

Summarizing, each ship moves totally independently and asynchronously from the others, not having any bound on the time it needs to perform a *Move*, hence a cycle (it has to be, however, finite by Assumption A1); therefore, a ship can be seen *while* it is moving; in addition, they are oblivious, and anonymous. Moreover, no one of the followers knows in advance the path that the leader will follow, nor can it derive it at run-time. Their only task is to observe where the leader and the other followers are, reach an agreement — without communicating — on how and where to form the pattern in the plane, and move to positions such that the flock is formed and maintained.

Adopting such "simple" units aims at understanding what kind of complex tasks can be achieved, and under which conditions (for a detailed discussion on this model and its motivations, refer to [10, 8, 9, 12]).

3 The Flocking Problem

In this section we give a formal definition of a family of problems that we call collectively the *Flocking Problem*. In particular, we propose two variants of the problem, and characterize through several metrics the degree of acceptability of an approximate solution.

3.1 Definitions

Definition 3.1 (Formation). A *formation* $\mathbb{F} = \{p_1, \ldots, p_{n-1}, p_L\}$ is a configuration with a distinguished point, p_L. We call the distinguished point the *leader* of the formation, and the remaining points the *followers* of the formation.

We call *radius* of a formation $\mathbb{F}$, denoted by $R_{\mathbb{F}}$ the maximum distance between the designated point p_L and the other points in $\mathbb{F}$:

$$R_{\mathbb{F}} = \max_{i=1\ldots n-1} dist(p_L, p_i). \qquad \lhd\rhd$$

Configurations are used to model the positions of a set of vehicles, and also to express the set of points that constitute the desired formation. The formation whose points are the current positions of the vehicles (included the leader) is called the (current) *fleet* (denoted by $\mathbb{E}$), while the formation given in input to the robots, and whose points represent the desired position of the vehicles once the flock is formed, is called the *pattern* (denoted by $\mathbb{P}$).

In order to assess the degree of success of the flocking, we need a measure of how well the current fleet approximate the desired pattern. We introduce such a measure in the following

Definition 3.1 ($\mathcal{D}$-distance). Given two configurations $C = \{c_1, \ldots, c_n\}$ and $G = \{g_1, \ldots, g_n\}$, we define the $\mathcal{D}$ distance among them as follows:

$$\mathcal{D}(C, G) = \min_{\pi \in \Pi} \sum_{i=1}^{|C|} dist(c_i, g_{\pi(i)})$$

where Π is the set of all the possible permutations of $1 \ldots |C|$.

Moreover, we define $\rho_i(C, G) = dist(c_i, g_{\pi'(i)})$, where π' is the permutation such that $\mathcal{D}(C, G) = \sum_{i=1}^{|C|} dist(c_i, g_{\pi'(i)})$. As a shorthand, we will write simply ρ_i when the configurations are evident in the context. $\lhd\rhd$

As an aside, we note that other kind of measures can be used. For instance, the maximum of the sum of the distances, or its average, or the sum of the squares of the distances. Our experiments, however, have been tested according to the measure defined in Definition 3.1.

We can consider that the desired pattern is reached when the vehicles place themselves in the desired shape, with no regard for the orientation, or we can ask, in addition, for a specific orientation (typically, corresponding to the current heading of the leader). These two alternatives are defined formally in the following

Definition 3.2 (Target). Given a pattern $\mathbb{P}$ and a fleet $\mathbb{E}$, we call an *undirected target* of the vehicles any formation that is obtained by translating $\mathbb{P}$ so that its leader point coincides with the leader of $\mathbb{E}$, and rotated by an arbitrary angle. We denote such a formation with $\mathcal{T}_{\mathbb{P},\mathbb{E}}$.

Given, in addition, an angle θ, we call the *directed target* of the vehicles the particular undirected target that is rotated by θ. We denote this formation with $\mathcal{T}^{\theta}_{\mathbb{P},\mathbb{E}}$ We call the followers' positions in a target the *slots* of the target. $\lhd\rhd$

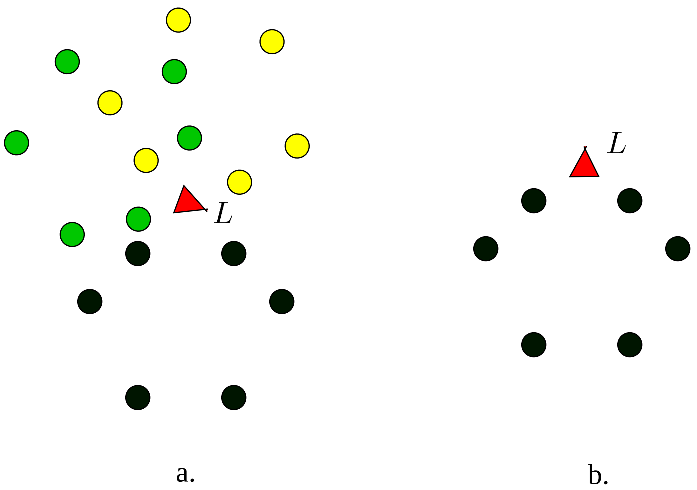

Figure 1: Undirected and directed targets. The triangle represents the leader.

Notice that, given a pattern and the position and heading of the leader, there are infinite undirected targets (Figure 1.a), but only one directed target (Figure 1.b) — that, naturally, is also an undirected target.

Since in our model the leader is constantly moving, while the followers only execute discrete cycles, it is impossible for them to exactly form and maintain the pattern at any desired time. To take this effect into account, we introduce two distinct notions for *exactly* and *approximately* forming the pattern in the definitions below.

Definition 3.2 (Exact Formation). Given a fleet $\mathbb{E}$ and a pattern $\mathbb{P}$, we say that the followers in $\mathbb{E}$ exactly form an undirected target $\mathcal{T}_{\mathbb{P},\mathbb{E}}$ if

$$\mathcal{D}(\mathbb{E}, \mathcal{T}_{\mathbb{P},\mathbb{E}}) = 0.$$

Moreover, if ψ is the heading of the leader, we say that the followers exactly form the desired pattern if

$$\mathcal{D}(\mathbb{E}, \mathcal{T}^{\psi}_{\mathbb{P},\mathbb{E}}) = 0. \quad \triangleleft\triangleright$$

We extend the above definitions to the case in which the formation is not kept exactly.

Definition 3.3 (Approximate Formation). An undirected target is formed up to ξ if

$$\mathcal{D}(\mathbb{E}, \mathcal{T}_{\mathbb{P},\mathbb{E}}) \leq \xi.$$

Analogously, the directed target is formed up to ξ if

$$\mathcal{D}(\mathbb{E}, \mathcal{T}^{\psi}_{\mathbb{P},\mathbb{E}}) \leq \xi. \quad \triangleleft\triangleright$$

Finally, we can introduce our formal definition of the Flocking Problem.

Definition 3.4 (The Flocking Problem). Let $f_1, \ldots, f_{n-1}$ be a group of vehicles according to our model, and let L be an additional distinguished leader vehicle, with heading ψ_L, whose positions constitute a formation $\mathbb{E}$, and let $\mathbb{P}$ be a pattern given in input to $f_1, \ldots, f_{n-1}$. The vehicles solve the exact (resp. approximate) *Flocking Problem* if, starting from an arbitrary formation at time t_0, $\exists t_1 \geq t_0$ such that, $\forall t \geq t_1$ the vehicles exactly (resp. up to ξ) form a certain formation. More specifically, four variants of the flocking problem exists:

- exact undirected flocking: $\mathcal{D}(\mathbb{E}, \mathcal{T}_{\mathbb{P},\mathbb{E}}) = 0$
- exact directed flocking: $\mathcal{D}(\mathbb{E}, \mathcal{T}^{\psi_L}_{\mathbb{P},\mathbb{E}}) = 0$
- approximate undirected flocking: $\mathcal{D}(\mathbb{E}, \mathcal{T}_{\mathbb{P},\mathbb{E}}) \leq \xi$
- approximate directed flocking: $\mathcal{D}(\mathbb{E}, \mathcal{T}^{\psi_L}_{\mathbb{P},\mathbb{E}}) \leq \xi$ ◁▷

The exact flocking variants cannot be solved in our model, since we assume that the leader moves continuously and arbitrarily, while the followers only have discrete opportunities for observing the position of the leader and adjust their course accordingly. Hence, while exact flocking can be considered as an ideal reference problem, in the following we will concentrate on the two variants of approximate flocking.

Notice however that, since we characterize through ξ the degree of approximation, and since we will give conditions that relate ξ to the features of the vehicles, an arbitrarily good approximation can be obtained.

3.2 Conditions

In order for the problem to be solvable, a number of conditions must be met. Let v_L and ω_L be the maximum linear and angular velocity of the leader, respectively, and let v_f be the maximum linear velocity of follower f. Firstly, the leader must not be too fast, otherwise the followers will not be able to maintain the formation. Formally,

$$v_L < \min_i v_{f_i}. \tag{1}$$

Moreover, the slots must not move too fast for the followers, as a consequence of the leader changing direction; thus, also the angular velocity of the leader must be limited:

$$v_L + \omega_L \cdot R_{\mathbb{P}} < \min_i v_{f_i}. \tag{2}$$

In fact, in the worst possible case the leader is moving away from a follower f while at the same time turning so that the tangential velocity of the points in $\mathcal{T}^{\theta}_{\mathbb{P},\mathbb{E}}$ that f is trying to reach is maximal (see Figure 2).

Secondly, in order to maintain the flocking up to ξ, the time spent in a *Move* by a follower must not be too long. Otherwise, the leader could change

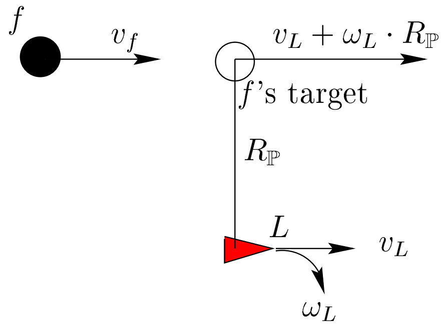

Figure 2: Limitations on the linear and angular velocity of the leader.

direction and move away from a follower in the time between two consecutive *Look*s, without the follower having a chance to correct its course. Formally, let

$$k^f = (v_L + \omega_L \cdot R_{\mathbb{P}}) \max_j \tau_f^M(j)$$

where $\tau_f^M(j)$ is the duration of the *Move* phase of the j-th cycle of the follower f. In the above definition, k^f is the maximum distance that a point in $\mathcal{T}^{\theta}_{\mathbb{P},\mathbb{E}}$ may travel during the longest *Move* phase of the follower f. Since we want that the overall $\mathcal{D}$-distance remains bounded by ξ, the following condition must be met:

$$\sum_i k^{f_i} \leq \xi \tag{3}$$

The condition above is overly restrictive, though, since we consider the maximum duration for the *Move* of all the followers. We can state a less restrictive (but still only sufficient) condition by considering the duration of a *Move* at a specified point in time. In detail, let $\bar{\tau}_f(t)$ be the time between t and the beginning of the *Move* of the follower f that is being executed at time t (see Figure 3). Then,

$$\forall t, \sum_i (v_L + \omega_L \cdot R_{\mathbb{P}}) \bar{\tau}_{f_i}(t) \leq \xi \tag{4}$$

As a last remark, we note that, since we want the vehicles to form a specific formation and *to keep that* formation while moving, it is necessary for the followers to agree on a common unit measure. Otherwise (i.e. if the input pattern can be scaled), the formation would be formed even if, instead of following the leader, the followers would simply "scale" the input formation (that is, we would have a situation where the vehicles would form a pattern that becomes bigger and bigger as the leader goes farther away, see Figure 4). Hence,

Observation 3.1. In order for the problem to be significant, $f_1 \ldots, f_{n-1}$ must have common knowledge on the unit measure. $\diamond$

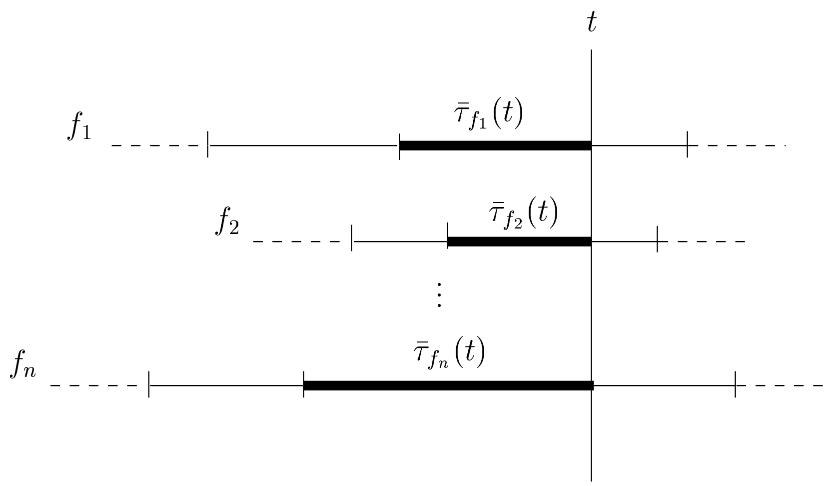

Figure 3: Definition of $\bar{\tau}_f$.

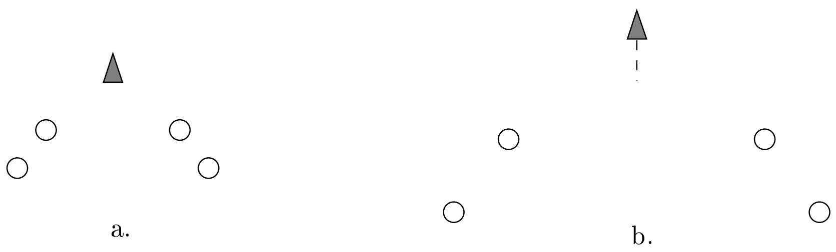

Figure 4: The triangle represents the leader, and the circles the positions of $\mathbb{P}$. If the vehicles did not have a common unit distance, then they could scale $\mathbb{P}$, and the flock would not follow L while it moves.

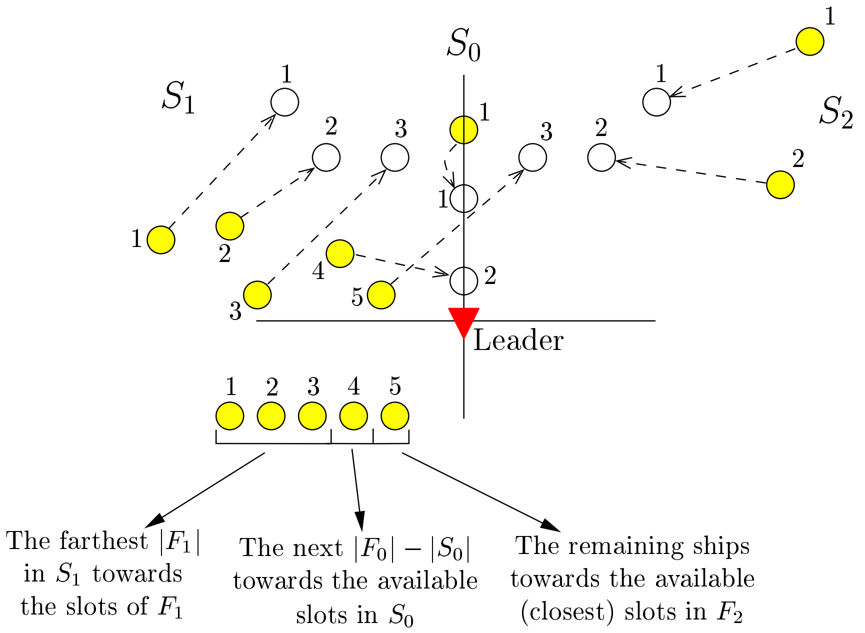

Figure 5: Examples of the behavior of Algorithm 1. The filled triangle is the leader, the filled circles are the followers, and the empty circles the *slots* the followers want to reach.

4 Basic Algorithm For The Flocking Problem

In this section we present an algorithm to solve the approximate directed flocking problem that works in the general setting where there is no agreement on the local coordinate systems. Every follower f_i is given in input a pattern $\mathbb{P}$ described as a set $p_1, \ldots p_{|\mathbb{P}|}$ of points, relatives to the leader vehicle, L; we clearly assume to have $|\mathbb{P}| - 1$ followers arbitrarily placed on distinct positions at the beginning (this defines a *valid* initial configuration for this problem).

The intuition behind the basic algorithm is described in the following (see also Figure 5). First, the generic follower f computes the baricenter B of the followers' positions (Line 1), by executing `Baricenter(`$Followers$`)` (*Followers* and L are the positions of the followers and of the leader retrieved in the previous *Look* state, respectively), and a shared vertical axis Y given by the line passing through L and B, and oriented according to $\overrightarrow{BL}$ can be derived: this is accomplished by `Get_Y_axis(`L, B`)`, that returns the axis Y that all the followers will use to agree on orienting themselves in the plane.[3] Then S_0, S_1 and S_2, containing respectively vehicles whose positions are exactly on Y, to its left, and to its right, are computed (according to the local concept of *left/right of*

[3]If $L = B$, the followers can simply wait for the leader to move away from B, or for some fellow follower that is already moving to break the tie.

Algorithm 1 The Basic Flocking Algorithm

Input: The pattern $\mathbb{P}$ to be kept, relative to the leader. $Followers$ and L are the positions of $f_1, \ldots, f_{n-1}$ and of the leader retrieved in the last *Look* state, respectively. me represents the robots executing the algorithm.

$B :=$ `Baricenter`$(Followers)$;
$Y :=$ `Get_Y_axis`(L, B);
$S_0 := \{$Robots On $Y\}$;
$S_1 := \{$Robots On The Left Of $Y\}$;
$S_2 := \{$Robots On The Right Of $Y\}$;
$F :=$ `Final_Positions`$(\mathbb{P}, L, Y)$;
$B_F :=$ `Baricenter`(F);
$F_0 := \{$Final Positions On $Y\}$;
$F_1 := \{$Final Positions On The Left Of $Y\}$;
$F_2 := \{$Final Positions On The Right Of $Y\}$;
For All $j = 0, 1, 2$ **Do**
    `Sort`(F_j, L, B_F);
    `Sort`(S_j, L, B);
End For
Case me in

- S_1
  $k :=$ `Rank`(me, S_1);
  If $k \leq |F_1|$ **Then**
      `Move`(k-th Position In F_1).
  Else If $k \leq |F_1| + |F_0| - |S_0|$ **Then**
      $H := \{$Robots In S_1 Whose Rank $> |F_1|\} \cup$
          $\{$Robots In S_2 Whose Rank $> |F_2|\}$;
      `Sort`(H, L, B);
      $k' :=$ `Rank`(me, H);
      $p := (k' + |S_0|)$-th *slot* in F_0;
      `Move`(p).
  Else
      `Move`($(|F_2| - (k - |F_1| - |F_0| + |S_0|) + 1)$-th Position In F_2).
- S_2
  /* This case is symmetric to the previous one */
- S_0
  $k :=$ `Rank`(me, S_0);
  If $k \leq |F_0|$ **Then**
      `Move`(k-th Position In F_0).
  Else If $|S_1| \leq |S_2|$ **Then**
      `Move`($(|F_1| - |S_1| + (k - |F_0|))$-th Position In F_1).
  Else
      `Move`($(|F_2| - |S_2| + (|k - |F_0||))$-th Position In F_2).

Y).

At this point, f executes `Final_Positions(`$\mathbb{P}, L, Y$`)` (Line 6), that rotates the points in $\mathbb{P}$, assuming that the leader is moving according to the direction and orientation of Y, and translates them into the observed leader's position. The positions returned by this routine are the *slots* that the followers will try to reach. After having computed the baricenter B_F of the *slots* in Line 7, these positions are partitioned in three subsets: those exactly on Y (F_0, Line 8), to the left of Y (F_1, Line 9), and to its right (F_2, Line 10). Then, F_j, $j = 0, 1, 2$, are sorted in decreasing order with respect to the distances from L and B_F (Line 12), and S_j, $j = 0, 1, 2$, are sorted in decreasing order with respect to the distances from L and B (Line 13). These sorting operations are done by `Sort(`A, l, b`)`, where A is the array (set of points) to be sorted. In particular, after the sorting, it is guaranteed that

$$\begin{aligned}\forall i, j,\ i < j \quad \Rightarrow \quad & (dist(l, p_i) > dist(l, p_j)) \vee \\ & (dist(l, p_i) = dist(l, p_j) \wedge dist(b, p_i) > dist(b, p_j)),\end{aligned}$$

where p_i and p_j are points in A. Next, the rank k of f in the subset it belongs to is computed, by `Rank(`$me, \cdot$`)`.

Now, if f is the k-th follower in S_1, and $k \leq |S_1|$, then it moves towards the k-th position in F_1 (Line 19; a similar argument applies if f is in S_2, see Line 34). Otherwise, if there are positions *available* in S_0 (i.e., $|F_0| > |S_0|$ and $k - |F_1| \leq |F_0| - |S_0|$), f is directed towards S_0. In particular, Line 21 computes the set H containing the vehicles in S_1 and S_2 whose rank is respectively bigger than $|F_1|$ and $|F_2|$; H is then sorted, and the rank k' of f in H is computed in Line 24. Then, f is directed towards the $(k' + |S_0|)$-th *slot* in F_0 (Lines 25–26), that is towards a *slot* in F_0 that is not a target of vehicles in S_0 (refer to Lines 31–38 to see how vehicles in S_0 choose their targets). If no position in S_0 is available, f moves towards the $(|F_2| - (k - |F_1| - |F_0| + |S_0|) + 1)$-th position in F_2, that is towards one of the *slots* in F_2 that are not a destination point of either a vehicle in S_1 whose rank is smaller than k, or of a vehicle in S_2 (Line 28).

If f is in S_0, and its rank k is smaller than $|F_0|$, then it simply moves towards the k-th *slot* in F_0 (Line 34). Otherwise, it chooses to move towards the side that has fewer vehicles (note that if $|S_1| = |S_2|$, then it chooses to move towards a slot in F_1). In Figure 5, an example of how the followers chose their *slots* is depicted.

Finally, `Move(`p`)` moves the executing vehicle towards p, and terminates the current cycle. As already pointed out, in general the vehicle does not reach p in one *Move* (the distance it travels is finite, but unpredictable). Clearly, since the vehicles can not remember p in the next cycle (obliviousness), this implies that it is possible that f changes its destination point in the next cycle, because its ranking can change.

Since the vehicles are memoryless, they can not be sure of the direction of movement of L. They only assume that the leader is going away from B (i.e. according to $\overrightarrow{BL}$). Furthermore, the followers assume that the direction of

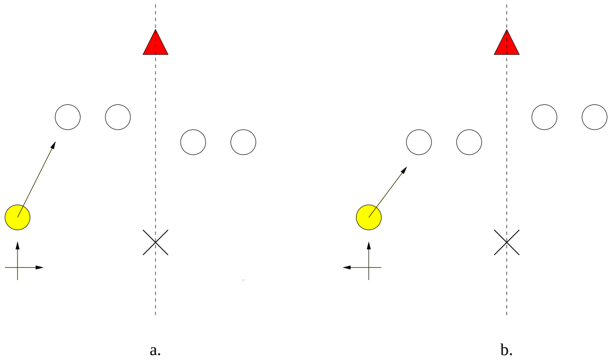

Figure 6: With a non-lb-symmetric pattern, the destination point of a unit depends on the chirality of the local coordinate system.

movement of L is given by the axis passing through B and L, and oriented from B towards L, hence they can reach an agreement on Y. They can not, however, reach in general a similar agreement on X, that is on an axis hortogonal to Y that would let them agree on the concept of *left* and *right*. Hence, the basic algorithm applies only to formations that are symmetric with respect to the direction of movement of L.

4.1 Applicability of The Algorithm

The basic algorithm statistically solves the approximate undirected flocking problem. Furthermore, it statistically solves the approximate directed flocking problem, provided that

1. the pattern is centered on the leader and directed according to the $\overrightarrow{BL}$ axis;
2. the pattern $\mathbb{P} = \{p_1, \ldots, p_n, L\}$ is symmetric with respect to the axis passing through L and the baricenter of the other points in $\mathbb{P}$ (we call this kind of patterns *lb-symmetric*), and
3. the pattern contains at most 1 point lying on this axis.

In fact, if the pattern is not lb-symmetric, the target *slot* of a follower robot depends on whether the follower is in S_1 or S_2, and thus on whether it should try to form the *semi-patterns* F_1 or F_2. However, this assignment depends on the chirality of the local coordinate system (see Figure 6), and thus the followers cannot generally reach an agreement on the formation to keep.[4] Notice that if the pattern is lb-symmetric, the choice still cannot be made, but it becomes unrelevant since the two semi-patterns are indistinguishable.

[4]In particular, no agreement can be reached if all the vehicles occupy positions that are symmetric with respect to the $\overrightarrow{BL}$ axis, while the pattern is not symmetric. See Section 6.4 for a more detailed discussion on this case and for possible solutions.

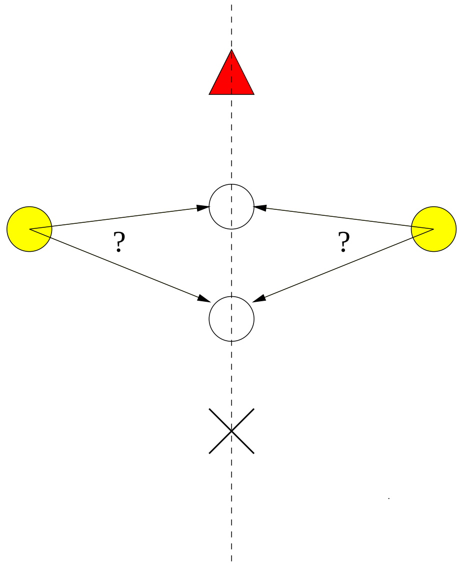

Figure 7: More than 1 position on the axis.

Also, if the pattern is lb-symmetric but has more than one position lying on the $\overrightarrow{BL}$ axis (see Figure 7), two vehicles could be ranked equal in step 24 of the algorithm. In this case, the two vehicles would select the same slot on F_0 as destination, preventing the correct formation of the pattern. Notice that for any number $m > 2$ of slots on the axis, the same problem could happen when $m - 2$ slots are taken by as many vehicles, while the two remaining vehicles could occupy symmetric positions as shown in Figure 7.

4.2 Analysis of Experiments

To measure how far the vehicles are at time t from the aimed formation, we use the following functions:

- $\Delta_e(t) = \mathcal{D}(\mathbb{E}(t), \mathcal{T}^{\phi_B(t)}_{\mathbb{P},\mathbb{E}(t)})$ the distance from the estimated formation, obtained from the position of the baricenter at time t. In particular, $\mathbb{E}(t)$ denotes the positions of the robots at time t, and $\mathcal{T}^{\phi_B(t)}_{\mathbb{P},\mathbb{E}(t)}$ is the directed target obtained by translating the leader of $\mathbb{P}$ onto the leader of $\mathbb{E}(t)$, and by rotating $\mathbb{P}$ of an angle $\phi_B(t)$ such that the baricenter of $p_1, \ldots, p_{n-1}$ lies on the line passing through the leader of $\mathbb{E}(t)$ and the baricenter of the followers in $\mathbb{E}(t)$; and
- $\Delta_r(t) = \mathcal{D}(\mathbb{E}(t), \mathcal{T}^{\psi_L(t)}_{\mathbb{P},\mathbb{E}(t)})$ the distance from the real formation, obtained taking into account the actual heading of the leader at time t, $\psi_L(t)$.

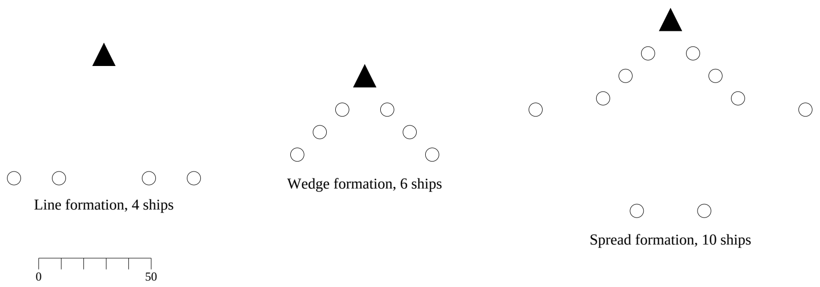

Figure 8: Fleet formations used in the simulations.

In Figure 11, some plots of Δ_e and Δ_r are reported, relative to computer simulations run with six followers trying to keep a wedge shaped formation, four in a line, ten in a spread formation (all shown in Figure 8), and with random formations.

In all cases, the simulations[5] started with all the vehicles (leader and followers) randomly placed in a square area with a side of 512 units. Each vehicle also had a random direction and orientation of the axes for its local coordinate system. Each follower had a random velocity between 0.5 and 5.0 units/move, while the leader's speed was determined in accordance with the limitations stated in Section 3.2. The fixed formations used in the experiments are shown in Figure 8; random formations were obtained by randomly choosing from two to eight symmetric points in the area delimited by the points (-150,-50) and (-50,+50), where (0,0) represents the leader and the axes are oriented so that x coincides with the leader's direction.

The leader's course was determined as follows: at all times, the leader would move forward according to its velocity. At each move, with a probability of 1/20, the leader could start turning to its left or right with an angular speed limited again according to Section 3.2. If already turning, with the same probability the leader could stop and continue its course as a straight line. Figures 9 and 10 show the courses of the vehicles in two simulation runs.

In Figure 11, it can be observed how convergence to the estimated formation (Δ_e) is obtained on average in less than half the time needed to reach the real formation. Figure 11 also reveals other interesting phenomena. In the Δ_r graph for the wedge formation, we can observe a "plateau" caused by the vehicles forming the pattern in exactly the wrong direction, i.e. in front of the leader rather than behind it. This correctly formed, but incorrectly-headed pattern, is kept until the leader starts changing its direction, at which point all the followers rapidly reach their proper positions behind the leader. Related effects can also be observed in the Δ_r graphs for the other formation, in which instabilities in the distance are caused by the followers trying to catch up with change of directions of the leader.

[5] The simulator has been written by Vincenzo Gervasi - University of Pisa.

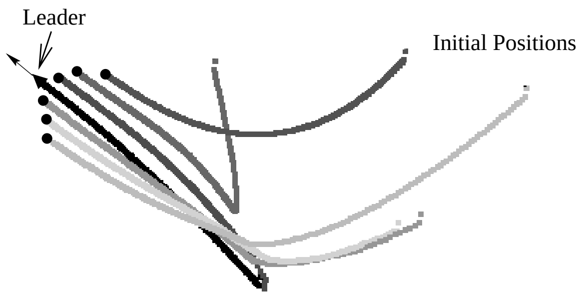

Figure 9: Trace of the vehicles while forming and keeping a wedge shaped formation.

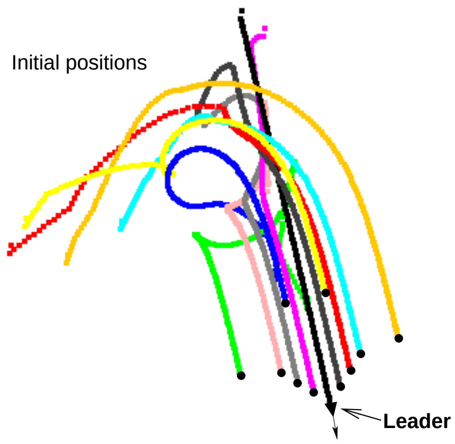

Figure 10: Trace of the vehicles while forming and keeping the spread formation with ten vehicles. Note the circular trajectory of the vehicles at the beginning, while trying to align the formation with the course of the leader.

All our experiments demonstrated that the algorithm is properly behaved, and in all cases the followers were able to assume the desired formation and to maintain it while following the leader vehicle along its route. Indeed, while Section 3.2 provides conditions under which the ability of the vehicles to follow the leader is guaranteed, even when those conditions were relaxed the followers were usually able to compensate for sudden turns or accelerations of the leader (as long as the effective speed of the leader remained, at least on average, lower than that of the slowest follower). As a further remark, we note that the obliviousness of the algorithm contributes to this result, since the followers do not base their computation on past leader's positions.

5 Extended Algorithms

The algorithm given above does not guarantee convergence, although simulations show that it provides statistical convergence in most cases. In this section we briefly discuss the problems with the basic algorithm, and provide two slightly more complex variations that solve these problems.

5.1 Problems With The Basic Algorithm

The basic algorithm suffers from a number of problems, and is subjected to somewhat restrictive conditions. In particular:

1. The basic algorithm converges rapidly only for approximate undirected flocking, while convergence in the case of approximate directed flocking is typically slower. This is caused by the followers' inability to observe the real heading of the leader (and by their obliviousness, since they cannot remember the previous position of the leader).

2. The followers can assume and maintain for an unpredictably long time a wrong formation. For instance, the followers can assume a formation that is specular to the correct one, and placed "in front" of the leader instead of behind it. In such a situation, as long as the leader maintains an heading that coincides with the $\overrightarrow{BL}$ axis, the followers will compensate any movement of the leader towards them by moving farther away, while keeping the formation on the wrong side and thus reproducing the same situation.

Problems 1 and 2 can be solved perfectly by observing the heading of the leader (i.e., by being able to distinguish the prow of the leader from the back), or — with a better approximation wrt the baricenter — by having enough memory to store the previous position of the leader.

3. In certain situations, two or more vehicles could continue changing the *slots* they have to reach, causing instability and slowing down or impeding altogether the convergence of the algorithm.

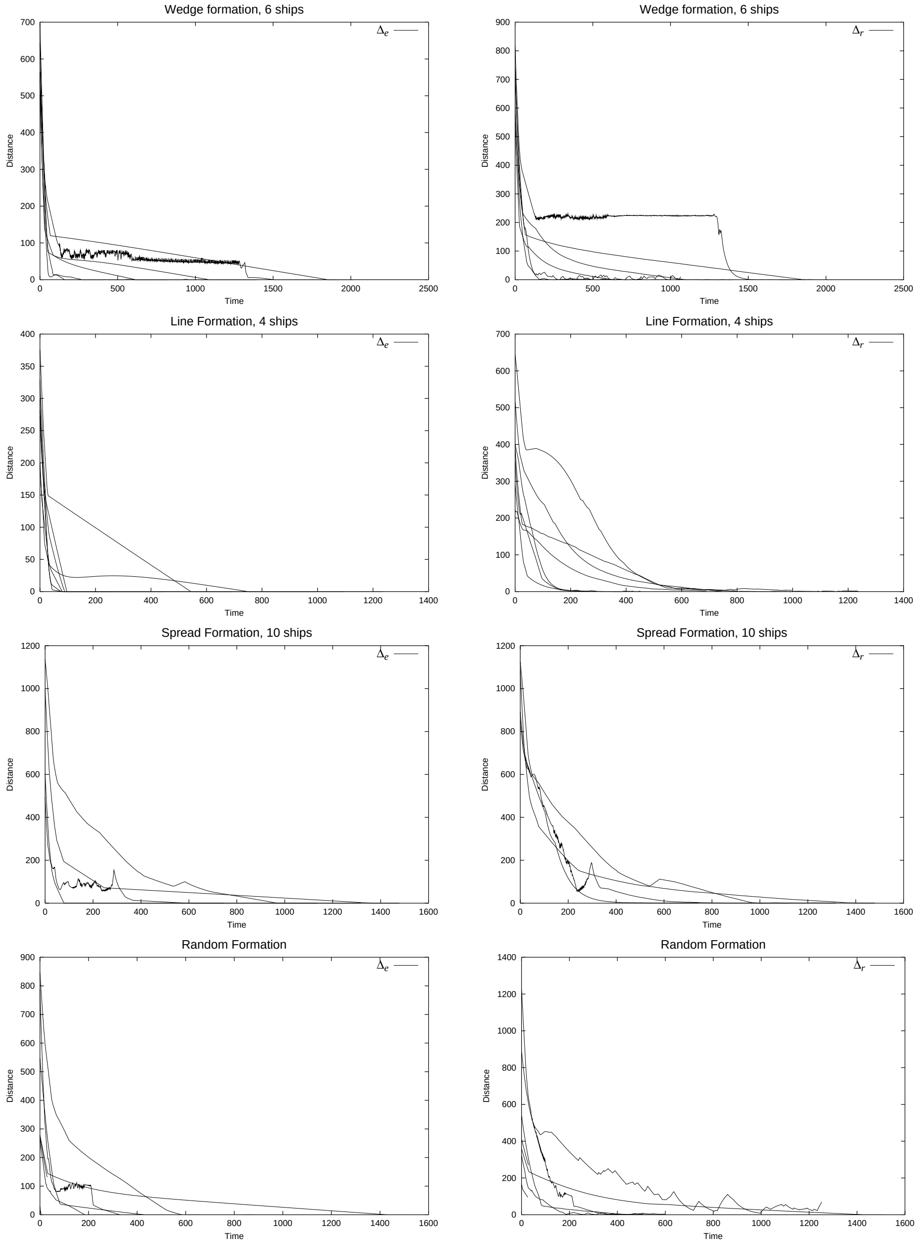

Figure 11: Some plot of the Δ_e and Δ_r, while forming a *wedge* of six vehicles, a *line* of four vehicles, a spread formation with ten vehicles, and random formations (with a number of vehicles variable between two and eight).

In the following, we discuss some suggestions that can help in fixing some of the problems pointed out with Algorithm 1. Unfortunately, we did not implement this ideas yet; hence, we can not present evidence of the goodness of such strategies. This will be part of our near future work.

5.2 The Hula-Hoop Algorithm

A source of instability in the basic algorithm lies in the fact that the ranking of the vehicles in a *semi-space* (S_1 or S_2) can change during the execution. As a consequence, the assignment of the slots to the vehicles can change, and this in turn can cause sudden changes of direction of the followers. Although very rare, it is also theoretically possible that, in pursuing the new slots, two followers keep exchanging their ranking, so that the flock never stabilizes.

One way to solve this problem consists in ensuring that the initial rankings among the followers never change during the execution of the algorithm.

In the basic algorithm, the ranking is assigned based on the lexicographical ordering that is given by the distance from the leader and from the baricenter. Let f, g and h be the $(i-1)$-th, i-th, and $(i+1)$-th vehicle in the ranking, respectively (refer to Figure 12). The area in which the ranking of g does not change is given by the region delimited by the circles having as center the leader L, and as radius $dist(L, f)$ and $dist(L, h)$ respectively (the obvious extensions apply if g is the first or last vehicle in the ranking). We call this region the *stable space* of g.

To try to maintain a stable ranking, each follower has to remain always in its stable space. This entails:

1. stopping before crossing the stable space boundary if the target is outside the stable space (waiting until the movements of other vehicles have changed the boundaries, possibly bringing the target inside the stable space), and

2. choosing curved trajectories instead of straight ones to reach a target that is inside the stable space when the straight trajectory would cross the boundaries.

While the above strategy increases the stability of the algorithm, it is not sufficient to guarantee it. In fact, it is possible that the movement of the leader, by changing the distances between the leader and the followers, changes the ranking of the followers even if they try to stay inside their stable spaces.

5.3 The Stripe Algorithm

A second variant of our basic algorithm increases the stability by changing the measure upon which the ranking is based rather than by changing the followers' strategies, as done with the hula-hoop algorithm.

The stripe variant uses a lexicographical ordering of the vehicles in a semi-space based on the distance from the Y axis, from the leader, and from the

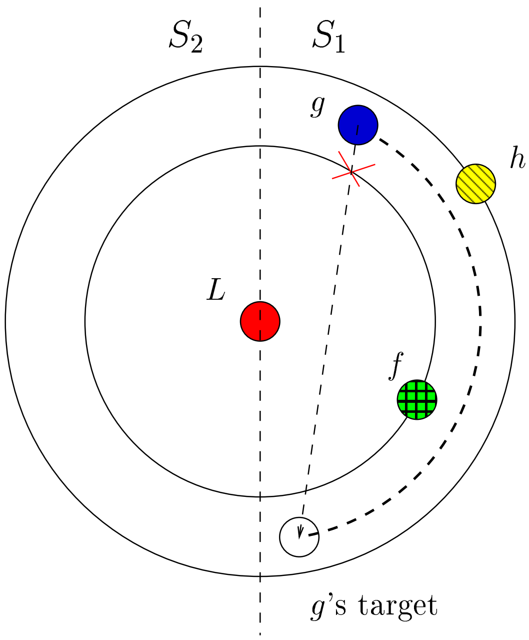

Figure 12: Strategy for the hula-hoop algorithm.

baricenter. More specifically, the `Sort(`A, l, b`)` routine in the basic algorithm is changed so that, after the sorting,

$$\begin{aligned}\forall i, j,\ i < j \quad \Rightarrow \quad & (dist(Y, p_i) > dist(Y, p_j)) \lor \\ & (dist(Y, p_i) = dist(Y, p_j) \land dist(l, p_i) > dist(l, p_j)) \lor \\ & (dist(Y, p_i) = dist(Y, p_j) \land dist(l, p_i) = dist(l, p_j) \land \\ & \qquad dist(b, p_i) > dist(b, p_j))\end{aligned}$$

where $dist(Y, p)$ is the distance between the Y axis (passing through l and b) and the point p.

In this case, the stable space of a follower is defined as follows (refer to Figure 13). Let f, g and h be the $(i-1)$-th, i-th, and $(i+1)$-th vehicle in the ranking, respectively. The area in which the ranking of g does not change is given by the stripe parallel to the Y axis and delimited by the lines, also parallel to the Y axis and passing through f and h, respectively (again, the obvious extensions apply if g is the first or last vehicle in the ranking).

Also in this case, the followers must take care not to deliberately cross their stable space boundaries. However, since the stripes are oriented according to the (estimated) direction of the leader, there is lesser risk that the movements of the leader can change the ranking and thus introduce instabilities. On the other hand, this variant is more sensible to variations of the *heading* of the leader, especially when the distance between the followers and the leader is

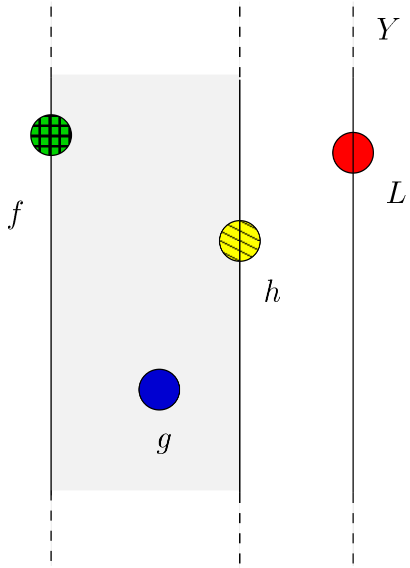

Figure 13: Stable spaces in the strip algorithm.

greater than a certain threshold, since the stripes can swipe rapidly. However, this effect is only significant if the *real* heading can be observed. When the heading is estimated via the baricenter, sudden change of estimated heading are possible only when the baricenter is near the leader, and this in turn typically only happens when the follower are near the leader, thus rendering the effects of the swipe less important.

Notice however that there is no need to follow non-rectilinear trajectories. This variant of the algorithm is thus simpler than the hula-hoop one.

6 Remarks on Different Levels of Common Knowledge

As can be expected, the amount of shared knowledge among the followers and their computational and observational capabilities condition the type of problems that can be solved. In this section, we analyze how different combinations of shared knowledge, observation abilities and amount of memory influence the kind of flocking problems that can be solved by the followers.

6.1 Observability

Our model already assures us that the positions of the leader and of the followers are observable by any follower, and all the algorithms presented so far only rely on this simple capability. If, in addition to that, we can also observe the heading of the leader (e.g., by observing the direction of its prow), we can avoid relying on the self-stabilization of a reference point (i.e., the baricenter) "behind" the leader, and lying along its direction of movement.

In particular, the followers can compute the *slots* to reach according exactly to the current direction of the leader, rather than approximating it based on the $\overrightarrow{BL}$ axis. Thus, it never happens that the followers stabilize on wrong directions (e.g., in front of the leader rather than behind it), and convergence is faster (actually, Δ_r coincides with Δ_e).

We do not discuss here whether being able to observe the direction of movement of the followers can improve[6] further the convergence.

6.2 Local Coordinate Systems

We distinguish four different levels of agreement on the direction and orientation of the coordinate axes.

6.2.1 No agreement

The basic algorithm we presented in Section 4 does not assume any agreement among the followers on the direction and orientation of the axes. In this case, we have that:

- The basic algorithm and its variants in Section 5 solve the approximate undirected flocking problem if the input pattern is lb-symmetric and contains at most 1 point lying on the $\overrightarrow{BL}$ axis.
- They also solve the approximate directed flocking problem, assuming that the leader does not indefinitely keep the same direction. In this case convergence is slower than in the previous one.
- If the pattern is lb-symmetric, but contains more than 1 point on the $\overrightarrow{BL}$ axis, it may not converge.
- *Any* deterministic algorithm may not converge if the pattern is not lb-symmetric.
- A non-deterministic algorithm may reduce the probability of non-convergence to an arbitrarily small amount (see discussion in Section 6.4)

6.2.2 One axis direction and orientation agreement

In the case in which the followers agree only the *direction* of a single axis[7] (and not on its orientation), they can use it and the $\overrightarrow{BL}$ axis to obtain a chirality (e.g., by assuming that the clockwise direction for angles goes from $\overrightarrow{BL}$ towards

[6]This would allow some estimation of where the other followers will be at some future time.
[7]Hence, they agree on the direction of both axes.

the acute angle between $\overrightarrow{BL}$ and the shared axis x). However, if one of the two shared axes coincides with the $\overrightarrow{BL}$ axis, no chirality can be obtained. Thus, in general, having agreement on one axis direction does not improve the capability of the vehicles to solve the flocking problem.

It should be noted, however, that even if the two axes coincide, the followers could simply wait (by executing a *Move* towards their current position) until a move by the leader or by a fellow follower breaks the tie.

Once a chirality is obtained, it can be used to establish a shared orientation on the common axis. Thus, having agreement on the orientation of the shared axis at the beginning of the computation does not improve the followers' capability to solve the problem.

6.2.3 Chirality

The followers can observe L and B, and assume that $\overrightarrow{BL}$ is a shared Y axis. Then, given the chirality, all the followers can agree that a shared X axis is oriented according to the clockwise direction and assume B as the origin. Thus, given the chirality a complete shared coordinate system can be established.

6.2.4 Two axes

If all the followers agree on the direction and orientation of both axes, any tie condition can be broken. In particular, the followers can form non lb-symmetric patterns, and they can also form patterns with more than one slot on the Y axis.

6.3 Memory

So far we have discussed the case of oblivious algorithms, i.e. the robots cannot use any memory to store information about previous observations or decisions taken. In the following, the consequences of allowing bounded storage capabilities are discussed.

In particular, if the vehicles can store 1 position (e.g., two real numbers), the heading of the leader can be inferred by storing the position of the leader at the time of the last observation and considering the movement vector to the position observed in the current cycle. In this case, what said in Section 6.1 about being able to observe the real heading of the leader applies.

It is interesting to observe that even with an unbounded amount of memory (non-oblivious algorithm), the followers cannot form patterns that are not lb-symmetric. In fact, consider the case in which the initial configuration of the vehicles is lb-symmetric, while the pattern to be formed is not. Given a follower v, we call its *buddy* v' the follower that occupies the position symmetric to that of v in the other semi-space. If, for each follower, its buddy has the same velocity and moreover their look-compute-move cycles are perfectly synchronized, i.e. they execute the algorithm according to a synchronous activation schedule, we end up in configurations that maintain the symmetry for an indefinitely long

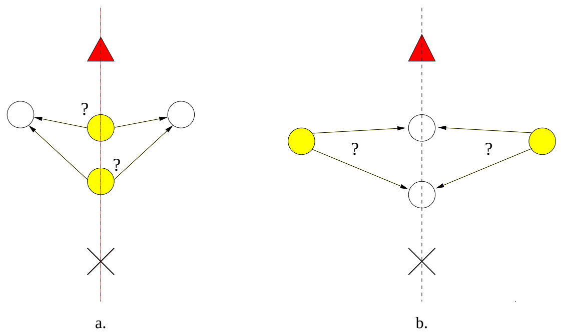

Figure 14: Cases of undeterminatedness with lb-symmetric formations with more than 1 point on the BL axis.

time.[8] Thus, any number of observations will not provide a means to break the symmetry, hence storing them does not help.

6.4 Randomization

If we can assume that the vehicles have the ability to take random choices (i.e., they are equipped with a proper entropy source), a larger class of problems can be solved. In particular, Algorithm 1 can be slightly modified in order to make it applicable also to formations that are not lb-symmetric or have more than one point lying on the $\overrightarrow{BL}$ axis.

In the latter case, two difficult cases arise, illustrated in Figure 14:

a. when two vehicles (or more) occupy positions lying exactly on the $\overrightarrow{BL}$axis, and two (or more) symmetrical slots are available, one on each semi-space. Lacking a shared direction for the x axis, the vehicles cannot agree on which of them should move towards the left-side slot and which towards the right-side one, and

b. when two vehicles (or more) occupy positions that are symmetric wrt the $\overrightarrow{BL}$axis, and two (or more) slots on the $\overrightarrow{BL}$axis are available. Lacking a shared direction for the x axis, the vehicles cannot agree on which of them should move towards which slot.

In such cases, our basic algorithm always chooses a fixed slot as target: towards the available slot in the *local* F_1 in case a., towards the slot closest to the leader in case b. Obviously, this strategy does not produce an assignment of vehicles to slots that covers all the slots. In practice, this "tie" conditions are usually broken by differences in the velocity of the vehicles and by the asynchronicity of their cycles.

However, a stronger guarantee can be obtained by randomly choosing the target slot among the possible candidates in steps 29 and 40 of the algorithm.

[8]Provided, naturally, that the algorithm employed by the followers is deterministic.

In the simplest case, involving only two slots, the probability of two vehicles always choosing the same target after k cycles is $1/2^k$. Even if the vehicles have the same velocity and synchronized cycles, as soon as they choose different target slots the symmetry is broken and the algorithm can continue normally.

Randomization can be used also in the case of non lb-symmetric formations. In this case, the algorithm for a follower f can

1. rank the semi-patterns according to some metric;
2. rank the semi-spaces according to the same metric;
3. assign the first semi-space to the first semi-pattern, and the second semi-space to the second semi-pattern;
4. rank the vehicles in the semi-space that contains f, and the slots in the corresponding semi-pattern;
5. choose a target for f in the semi-pattern corresponding to the semi-space that contains f, as done in the basic algorithm.

One way to compare configurations for the purpose of ranking semi-patterns and semi-spaces is defined by the following procedure:

`Ranking`
Input: Two configurations C and D with $|C| = |D|$, and two points L and B.
Output: $+1$ if C ranks higher than D, 0 if they rank equal, -1 otherwise.
`Sort`(C, L, B);
`Sort`(D, L, B);
$i := 1$;
While $i \leq |C|$ **Do**
  If $(dist(c_i, L) > dist(d_i, L)) \vee (dist(c_i, L) = dist(d_i, L) \wedge dist(c_i, B) > dist(d_i, B))$ **Then**
    Return $+1$;
  If $(dist(c_i, L) < dist(d_i, L)) \vee (dist(c_i, L) = dist(d_i, L) \wedge dist(c_i, B) < dist(d_i, B))$ **Then**
    Return -1;
  $i := i + 1$;
End While
Return 0;

Notice that the semi-patterns will always rank differently (since the pattern is asymmetric). It can happen, however, that the semi-spaces compare equals, that is, all the vehicles occupy positions that are symmetric wrt the $\overrightarrow{BL}$axis. In this case, the vehicles cannot agree on the assignment of semi-spaces to semi-pattern. Vehicles observing such a situation can, however, simply make a *Move* towards a random point in order to break the tie. As in the previous case, the probability of a tie continuing after k decisions is $1/2^k$.

7 Conclusions

In this paper we have presented a novel algorithm for coordinating a set of non-communicating, asynchronous and memoryless vehicles into following a leader while keeping a fixed formation. The algorithm only assumes the vehicles share a common unit of distance, but no common sense of direction (i.e., a common coordinate system) is needed, nor any previous knowledge of the path the leader will follow. Moreover, the followers do not need to have an identity, and are not distinguishable in any way one from the other.

Indeed, the algorithm we propose exhibits remarkable robustness, and numeric simulations indicate that in most cases the formation is reached in a relatively short time and kept after that, as desired.

References

[1] R. C. Arkin. Motor Schema-Based Mobile Robot Navigation. *Int. J. of Robot. Res.*, 8(4):92–112, 1989.

[2] R. C. Arkin and T. Balch. *Cooperative Multiagent Robotic Systems*, chapter in *Artificial Intelligence and Mobile Robots.* MIT/AAAI Press, D. Kortenkamp and R. P. Bonasso and R. Murphy edition, 1998.

[3] T. Balch and R. C. Arkin. Behavior-based Formation Control for Multi-robot Teams. *IEEE Trans. on Robot. and Autom.*, 14(6), 1998.

[4] Y. U. Cao, A. S. Fukunaga, A. B. Kahng, and F. Meng. Cooperative Mobile Robotics: Antecedents and Directions. In *IEEE/TSJ Int. Conf. on Intell. Robots and Sys.*, pages 226–234, 1995.

[5] Q. Chen and J. Y. S. Luh. Coordination and Control of a Group of Small Mobile Robots. In *Proc. of IEEE Int. Conf. on Robot. and Autom.*, pages 2315–2320, 1994.

[6] B. R. Donald, J. Jennings, and D. Rus. Analyzing Teams of Cooperating Mobile Robots. Technical Report TR 94-1429, Dept. of Comp. Sci., Cornell University.

[7] E. H. Durfee. Blissful Ignorance: Knowing Just Enough to Coordinate Well. In *International Conference on MultiAgent Systems (ICMAS)*, pages 406–413, 1995.

[8] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In *10th International Symposium on Algorithm and Computation (ISAAC '99)*, pages 93–102, 1999.

[9] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed Coordination of a Set of Autonomous Mobile Robots. In *IEEE Intelligent Vehicles Symposium (IVS2000)*, pages 480–485, 2000.

[10] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Asynchronous Mobile Robots with Limited Visibility. In *18th International Symposium on Theoretical Aspects of Computed Science (STACS 2001)*, 2001.

[11] V. Gervasi and G. Prencipe. Need a Fleet? Use The Force! In *FUN With Algorithms 2 (FUN 2001)*, pages 149–164, May 2001.